



## DevOps community Build you own CI

October 2018 – Cyril Marin & René Ribaud

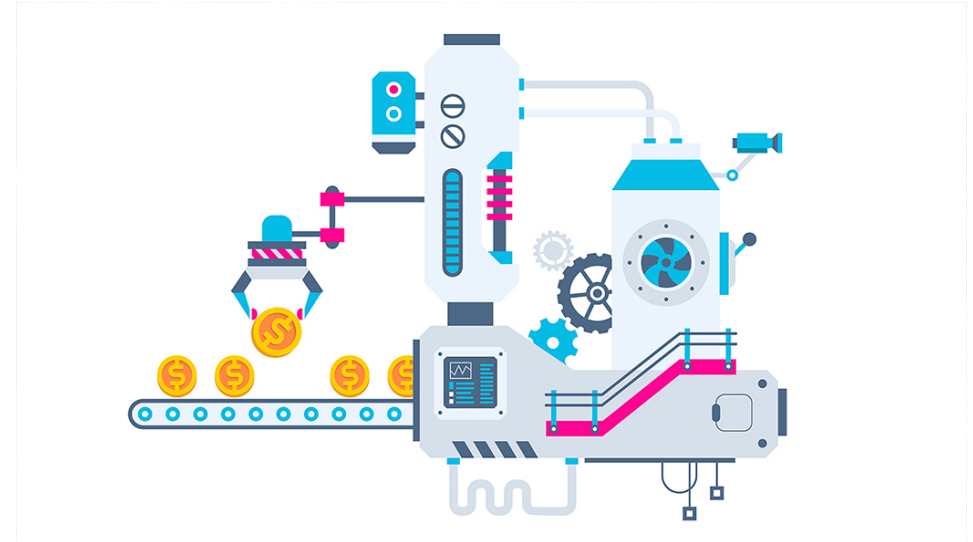
© CGI Group Inc. 2018

# CGI

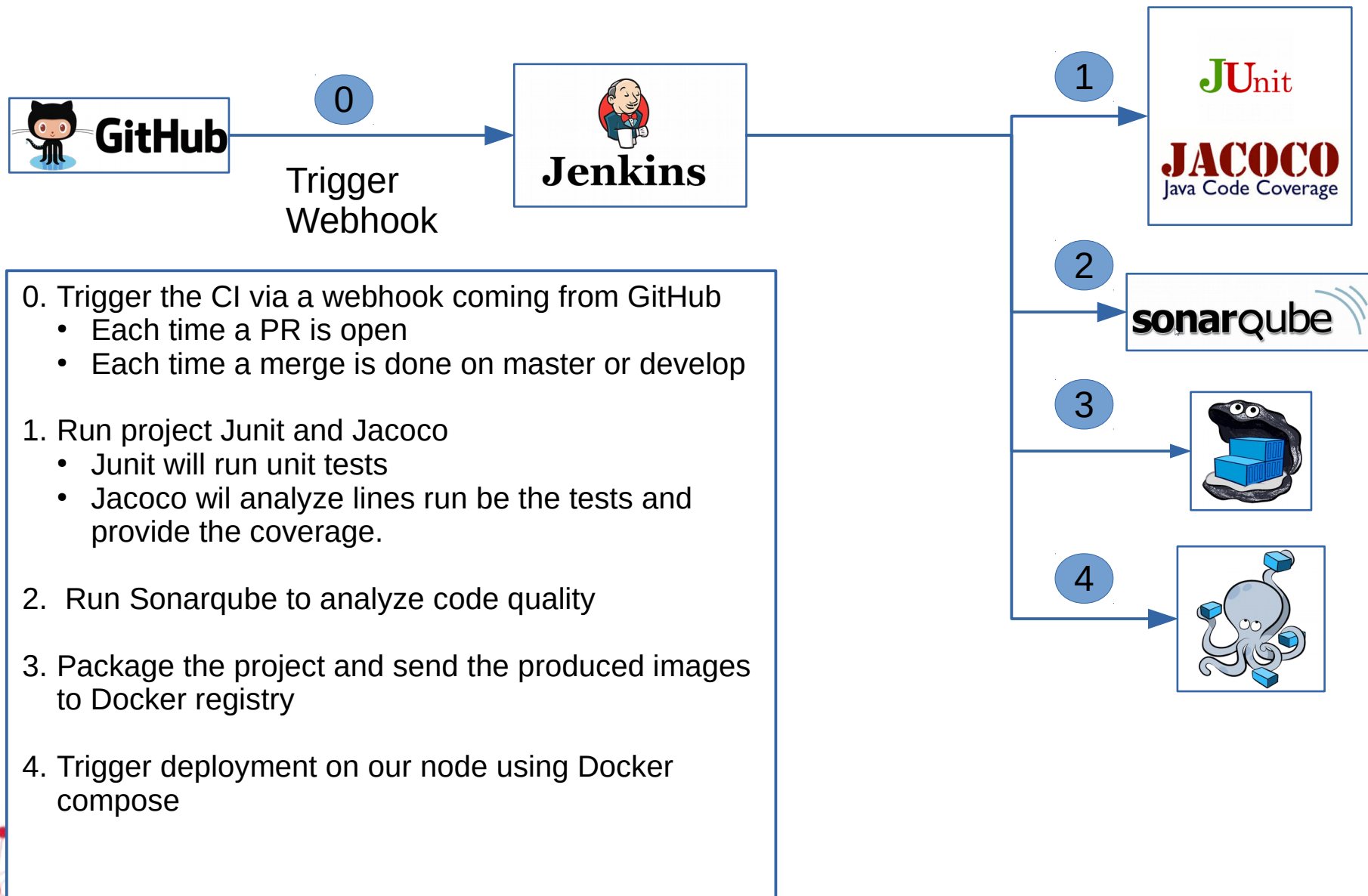
Experience the commitment®

# Goal of the session

- Build a simple Continuous Integration for a “hello world project”.
  - Capitalize on our Docker knowledge, tools will be installed using Docker images
  - Infrastructure used:
    - Project code will be hosted on Github
    - 1 x VM will host CI tools
    - 1 x VM will be used to deploy the project.



# CI/CD Pipeline



## Deploy 2 x VM

- Use the portal at [registry.uggla.fr](https://registry.uggla.fr) to create 2 Vms
  - Use docker 18.04 images
- In the next part of the document we will refer to the VM as
  - vmci → VM that will contains tools, warning **this VM should be created with 4GB of memory.**
  - vmdev → VM that will be used to deploy the project



# THE project !

Simple HelloWorld project

- URL : <https://github.com/uggla/HelloWorld>
- Fork the project using Github button  
To create your own project
- Clone your own project on the vmdev
  - `git clone <your project url>`
- Build and run the fantastic project using maven docker image  
(note maven images contains openjdk)
  - `docker run --rm -it -v /tmp/.m2:/root/.m2 -v $(pwd):/usr/src/myproject -w /usr/src/myproject maven:3-jdk-8-slim mvn package`
  - `docker run --rm -it -v /tmp/.m2:/root/.m2 -v $(pwd):/usr/src/myproject -w /usr/src/myproject maven:3-jdk-8-slim java -jar HelloWorld-0.1.0.jar`



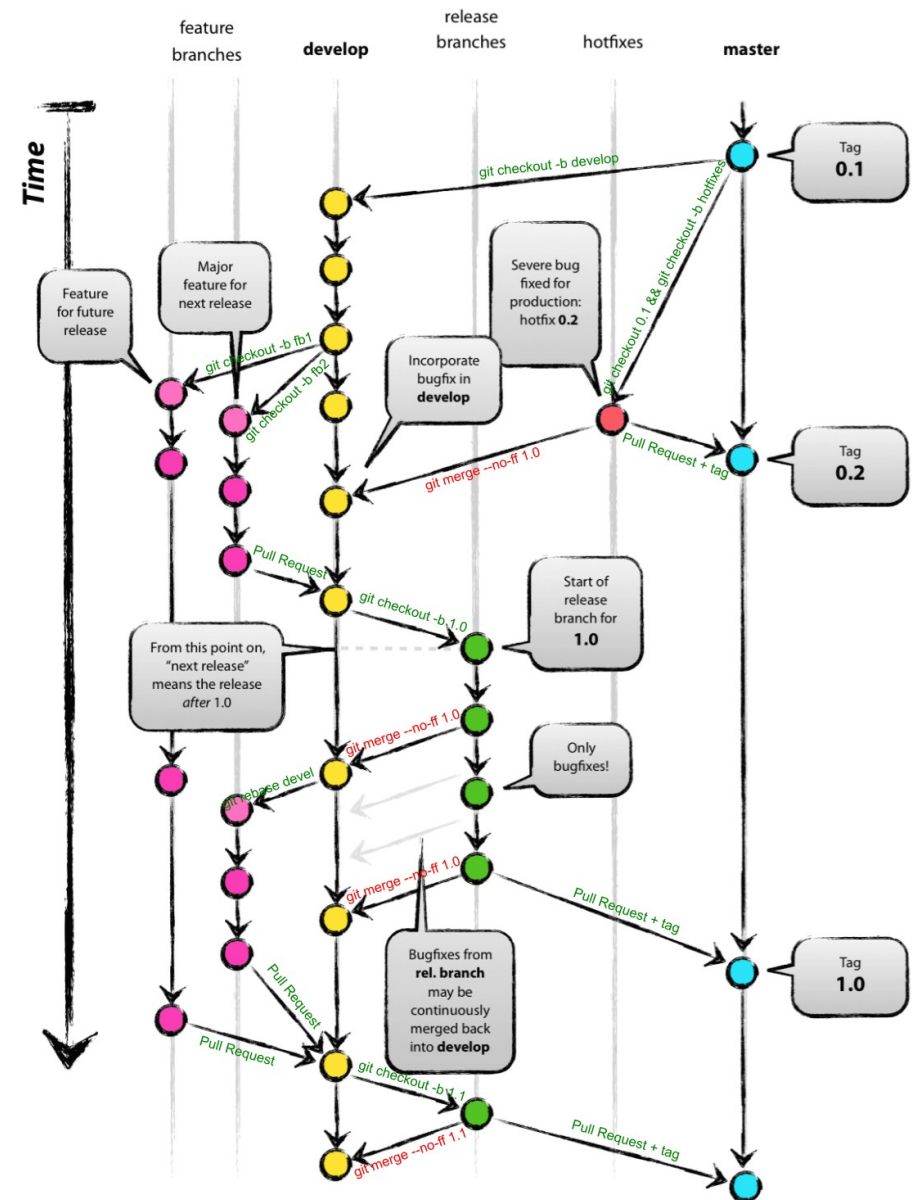
CGI



# Git branching model

This project will be managed using Git branching model, explaining this model in depth is not part of this session.

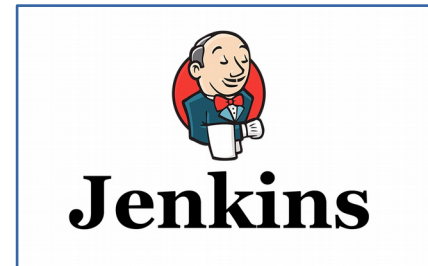
- Create the required develop branch
  - `git checkout -b develop`





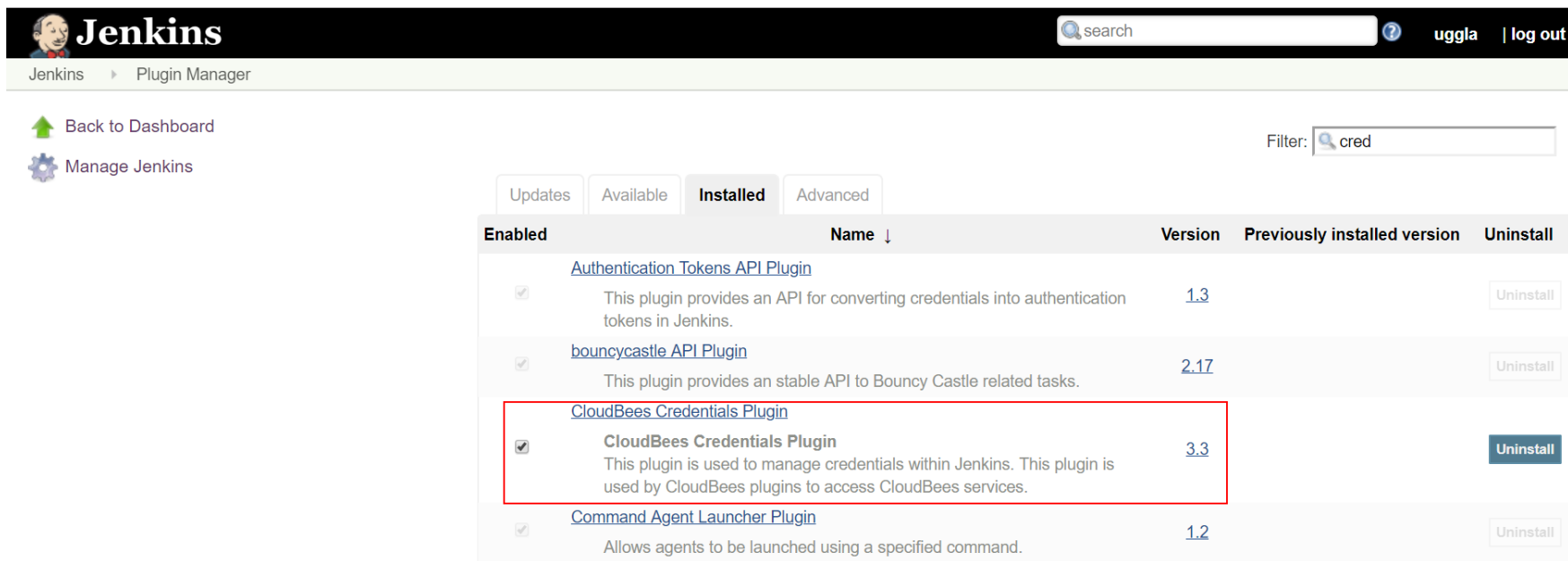
# Install the tools

- Install Jenkins
- Install Sonar
- To do it:
  - Go into project ci directory
  - Run run.sh
    - Create appropriate user for Jenkins
    - Set correct permission on the Docker socket
    - Install registry certificate from registry.uggla.fr



# Configuration Jenkins #1

- Go to `http://<vmci ip>:8080`
- Go ahead with the setup by installing recommended plugins
- Create an admin user, **please set a strong password as the vm is exposed to the internet**
- Install Cloubees credentials plugins + blue oceancd



The screenshot shows the Jenkins web interface. At the top is the Jenkins logo and a search bar. Below the logo, there's a breadcrumb trail: Jenkins > Plugin Manager. On the left sidebar, there are links for 'Back to Dashboard' and 'Manage Jenkins'. The main content area displays a list of installed plugins under the 'Installed' tab. The list is filtered by 'cred'. The plugins listed are: Authentication Tokens API Plugin (version 1.3), bouncycastle API Plugin (version 2.17), CloudBees Credentials Plugin (version 3.3), and Command Agent Launcher Plugin (version 1.2). The 'CloudBees Credentials Plugin' row is highlighted with a red border. Each row includes a checkbox for enabling/disabling, a description, the version number, and an 'Uninstall' button.

Enabled	Name ↓	Version	Previously installed version	Uninstall
<input checked="" type="checkbox"/>	<a href="#">Authentication Tokens API Plugin</a> This plugin provides an API for converting credentials into authentication tokens in Jenkins.	<a href="#">1.3</a>		<a href="#">Uninstall</a>
<input checked="" type="checkbox"/>	<a href="#">bouncycastle API Plugin</a> This plugin provides an stable API to Bouncy Castle related tasks.	<a href="#">2.17</a>		<a href="#">Uninstall</a>
<input checked="" type="checkbox"/>	<a href="#">CloudBees Credentials Plugin</a> <b>CloudBees Credentials Plugin</b> This plugin is used to manage credentials within Jenkins. This plugin is used by CloudBees plugins to access CloudBees services.	<a href="#">3.3</a>		<a href="#">Uninstall</a>
<input checked="" type="checkbox"/>	<a href="#">Command Agent Launcher Plugin</a> Allows agents to be launched using a specified command.	<a href="#">1.2</a>		<a href="#">Uninstall</a>





# Configuration Jenkins #2

- Configure the trigger
  - Create a personal access token to your Github account
  - Configure a Github server within Jenkins

Settings / Developer settings

OAuth Apps

GitHub Apps

Personal access tokens

## Edit personal access token

If you've lost or forgotten this token, you can regenerate it, but be aware that any scripts or applications using this token will need to be updated.

Regenerate token

### Token description

jenkins

What's this token for?

### Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

- |   |                                      |
|---|--------------------------------------|
| <input checked="" type="checkbox"/> repo            | Full control of private repositories |
| <input checked="" type="checkbox"/> repo:status     | Access commit status                 |
| <input checked="" type="checkbox"/> repo_deployment | Access deployment status             |

Jenkins > configuration

System Admin e-mail address

address not configured yet <nobody@nowhere>

### Lockable Resources Manager

Lockable Resources

Add Lockable Resource

### GitHub

GitHub Servers

#### GitHub Server

Name

GH

API URL

https://api.github.com

Credentials

mykeyforGH

Add

You can create your own [personal access token](#) in your account GitHub settings. Token should be registered with scopes:

- **admin:repo\_hook** - for managing hooks (read, write and delete old ones)
- **repo** - to see private repos
- **repo:status** - to manipulate commit statuses

In Jenkins, create credentials as «Secret Text», provided by [Plain Credentials Plugin](#).

**WARNING!** Credentials are filtered on changing custom GitHub URL.

If you have an existing GitHub login and password you can convert it to a token automatically with the help of «[Manage additional GitHub actions](#)».

(from [GitHub plugin](#))

Test connection

Manage hooks

☒

GitHub client cache size (MB)

20

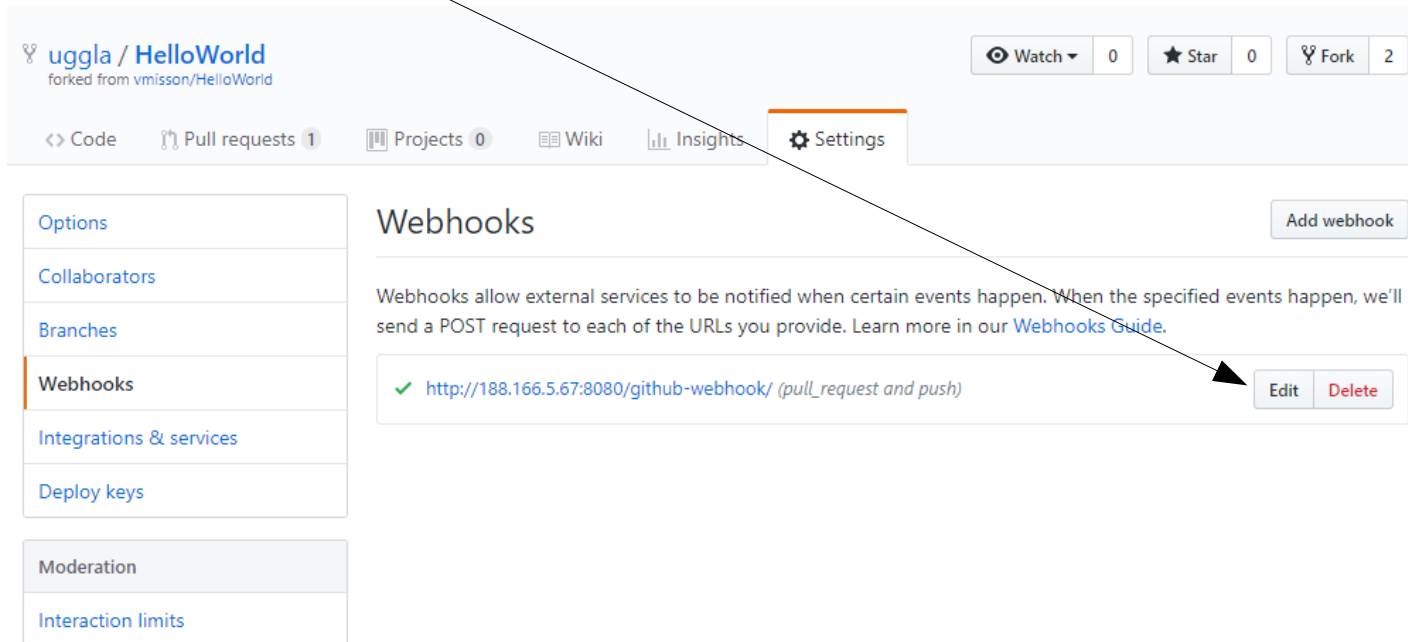
Delete



CGI

# Configuration Jenkins #3

- Configure the trigger
  - Check the Github repository webhook configuration
  - Click the edit button in order to have details and checks

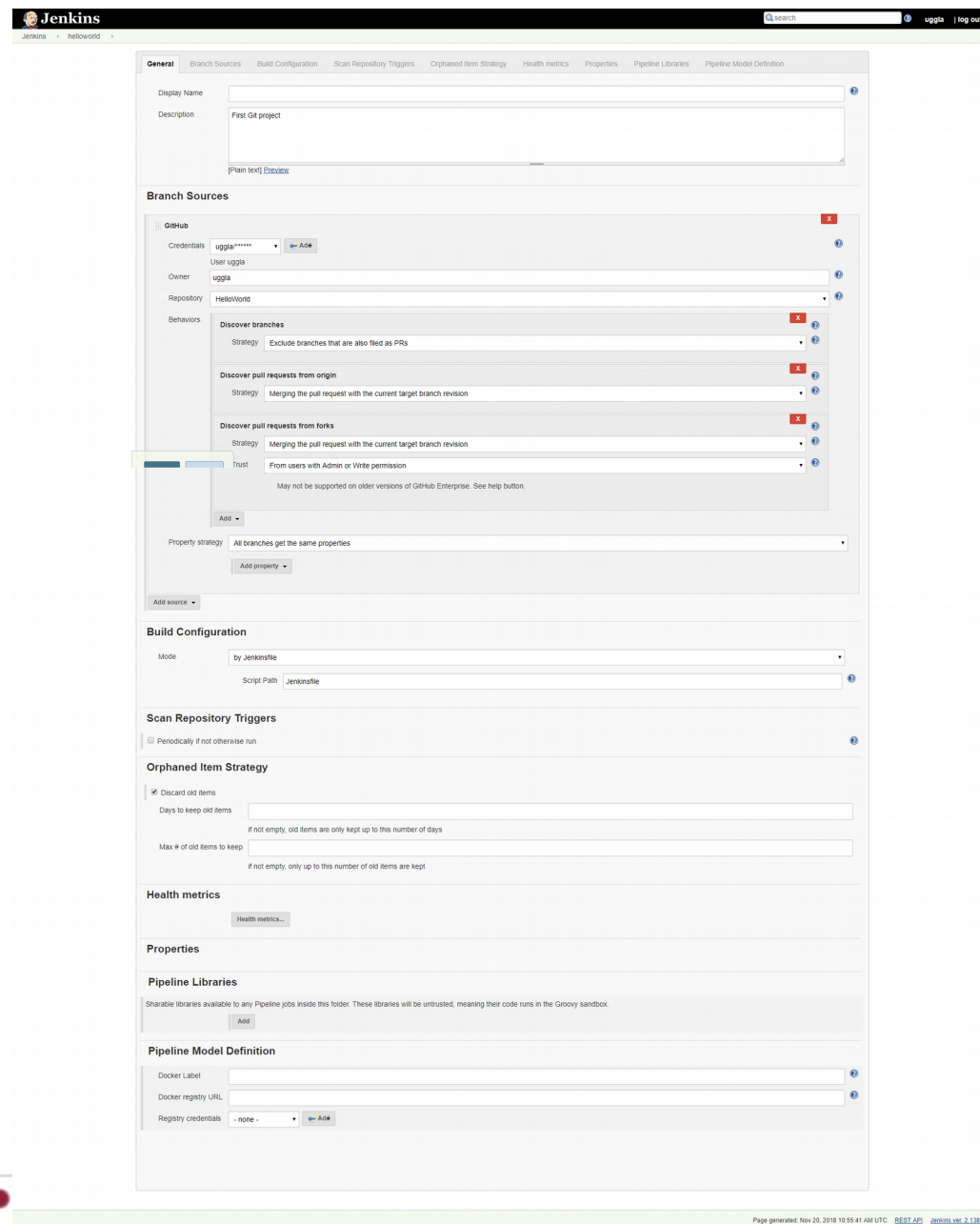


The screenshot shows the GitHub repository settings page for 'uggla / HelloWorld', which is forked from 'vmlsson/HelloWorld'. The repository has 0 watches, 0 stars, and 2 forks. The 'Settings' tab is selected in the top navigation bar. On the left sidebar, the 'Webhooks' section is highlighted. The main content area is titled 'Webhooks' and includes an 'Add webhook' button. Below this, a description states: 'Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).' A single webhook is listed with a green checkmark, the URL 'http://188.166.5.67:8080/github-webhook/' (with a note '(pull\_request and push)'), and 'Edit' and 'Delete' buttons. An arrow from the 'Click the edit button' instruction in the list points to the 'Edit' button.



# Configuration Jenkins #4

- Create a multibranch pipeline
  - Set the correct Github credentials
  - Set the correct repository
  - Set options as default settings



The screenshot shows the Jenkins configuration page for a multibranch pipeline. The top navigation bar includes the Jenkins logo, a search bar, and links for 'uggle' and 'log out'. The main configuration area is divided into several sections:

- General:** Includes fields for 'Display Name' and 'Description' (set to 'First Git project').
- Branch Sources:** Contains a 'GitHub' section with fields for 'Credentials' (set to 'uggle'), 'User' (set to 'uggle'), 'Owner' (set to 'uggle'), and 'Repository' (set to 'HelloWorld'). Below this are three 'Behaviors' sections: 'Discover branches', 'Discover pull requests from origin', and 'Discover pull requests from forks', each with a 'Strategy' dropdown. A 'Trust' checkbox is also present.
- Property strategy:** A dropdown menu set to 'All branches get the same properties'.
- Build Configuration:** Includes a 'Mode' dropdown set to 'by Jenkinsfile' and a 'Script Path' field set to 'Jenkinsfile'.
- Scan Repository Triggers:** A checkbox for 'Periodically if not otherwise run'.
- Orphaned Item Strategy:** Includes a 'Discard old items' checkbox and fields for 'Days to keep old items' and 'Max # of old items to keep'.
- Health metrics:** A 'Health metrics...' button.
- Properties:** A section for adding properties.
- Pipeline Libraries:** A section for adding shareable libraries.
- Pipeline Model Definition:** Includes fields for 'Docker Label', 'Docker registry URL', and 'Registry credentials' (set to 'none').

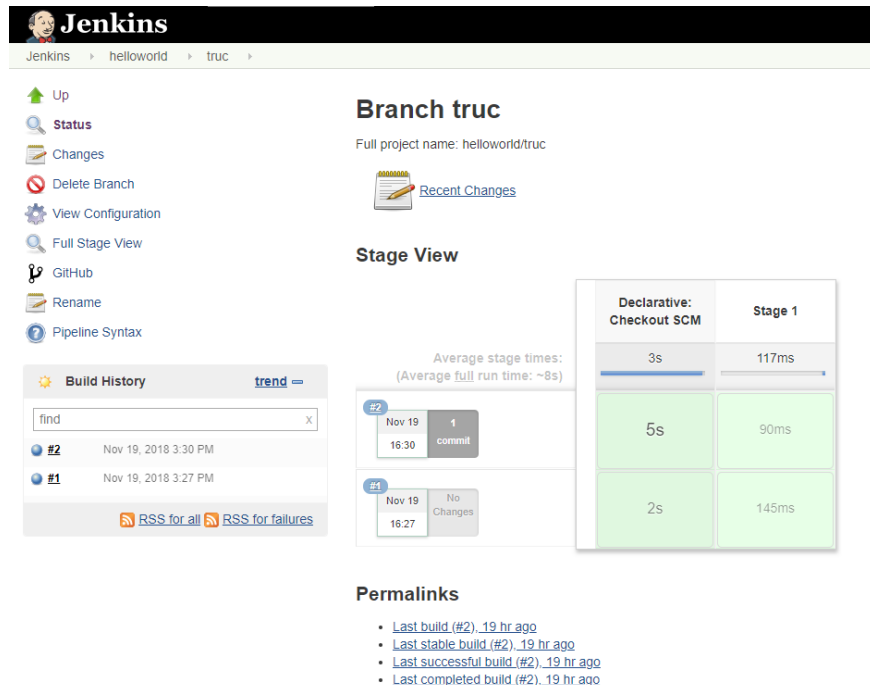
The bottom of the page shows a footer with the text 'Page generated: Nov 20, 2018 10:55:41 AM UTC' and links for 'REST API' and 'Jenkins ver. 2.138.3'.

# Define the pipeline #1

- Check CI
  - Create a small Jenkinsfile at the root of the project
- Commit and push the file
- A job should be successfully run and logs must contain the echos from Jenkinsfile
- If it works, create a new branch and an associated PR.

```
pipeline {
  agent any

  stages {
    stage('Stage 1') {
      steps {
        echo 'Hello world!'
        echo 'Coucou !'
        echo 'Welcome !'
      }
    }
  }
}
```



The screenshot shows the Jenkins web interface for a pipeline named 'truc'. The top navigation bar includes 'Jenkins', 'helloworld', and 'truc'. A left sidebar contains links for 'Up', 'Status', 'Changes', 'Delete Branch', 'View Configuration', 'Full Stage View', 'GitHub', 'Rename', and 'Pipeline Syntax'. The main content area is titled 'Branch truc' and shows the full project name 'helloworld/truc'. Below this, there's a 'Recent Changes' section with a table of build history. The 'Stage View' section displays a table of stage times for 'Declarative: Checkout SCM' and 'Stage 1'. The 'Permalinks' section provides links to the last build, last stable build, last successful build, and last completed build.

Build	Time	Status
#2	Nov 19, 2018 3:30 PM	1 commit
#1	Nov 19, 2018 3:27 PM	No Changes

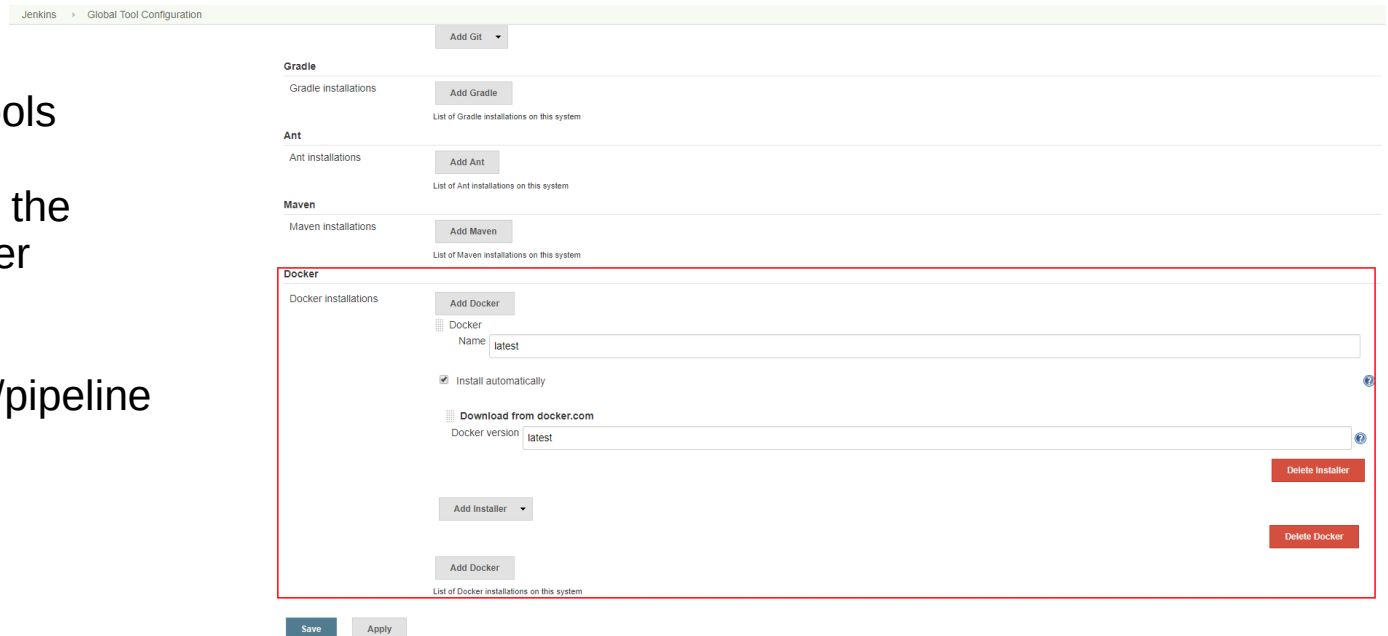
Declarative: Checkout SCM	Stage 1
3s	117ms
5s	90ms
2s	145ms

Permalinks

- [Last build \(#2\) 19 hr ago](#)
- [Last stable build \(#2\) 19 hr ago](#)
- [Last successful build \(#2\) 19 hr ago](#)
- [Last completed build \(#2\) 19 hr ago](#)

# Define the pipeline #2

- Build the project
  - Add Docker to the list of tools
- Update Jenkinsfile to build the project using maven Docker container  
Doc :  
<https://jenkins.io/doc/book/pipeline/docker/>



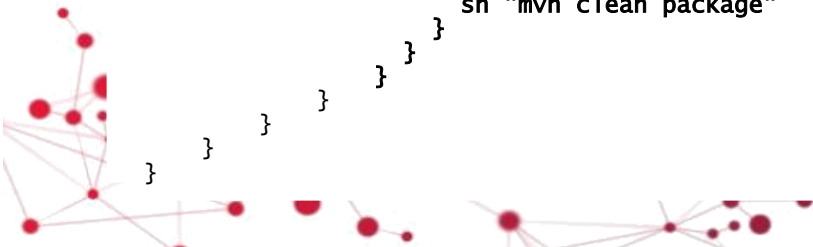
The screenshot shows the Jenkins 'Global Tool Configuration' page. The 'Docker' section is highlighted with a red border. It contains the following fields and controls:

- Gradle**: 'Add Gradle' button, 'List of Gradle installations on this system'.
- Ant**: 'Add Ant' button, 'List of Ant installations on this system'.
- Maven**: 'Add Maven' button, 'List of Maven installations on this system'.
- Docker**:
  - 'Add Docker' button.
  - 'Docker Name' text input field with 'latest' entered.
  - ☒ 'Install automatically' checkbox.
  - 'Download from docker.com' button.
  - 'Docker version' text input field with 'latest' entered.
  - 'Delete Installer' button.
  - 'Delete Docker' button.
  - 'Add Installer' dropdown menu.
  - 'Add Docker' button.
  - 'List of Docker installations on this system'.

At the bottom of the page are 'Save' and 'Apply' buttons.

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Hello world!'
                echo 'Coucou !'
                echo 'welcome !'
                script {
                    docker.withTool('latest') {
                        docker.image('maven:3-jdk-8-slim').inside("-v $WORKSPACE:/usr/src/myproject:rw -v $HOME/.m2:/root/.m2:rw -w /usr/src/myproject") { c ->
                            sh "mvn clean package"
                        }
                    }
                }
            }
        }
    }
}
```



# Add Sonar quality check #1

- Configure sonar
  - Connect to sonar `http://<vmci ip>:9000`
  - Login with `admin:admin` and set a token for your project (keep the token somewhere it will be used later)
  - Set a strong password as the vm is internet facing
  - Install sonarqube scanner for Jenkins plugin



# Add Sonar quality check #2

- Configure sonar
  - Configure the plugin

Jenkins » configuration

Lockable Resources  
Credentials  
New View

**Build Queue** —  
No builds in the queue.

**Build Executor Status** —  
1 Idle  
2 Idle

Labels  
Usage: Use this node as much as possible  
Quiet period: 5  
SCM checkout retry count: 0

☐ Restrict project naming

**Global properties**  
☐ Disable deferred wipeout on this node  
☐ Environment variables  
☐ Tool Locations

**SonarQube servers**

Environment variables  
☐ Enable injection of SonarQube server configuration as build environment variables  
If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

SonarQube installations

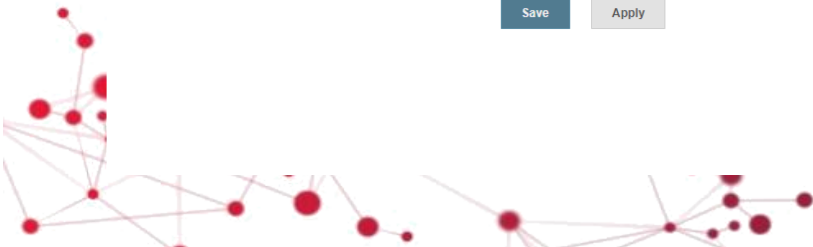
Name: sonar  
Server URL: http://sonarqube:9000  
Default is http://localhost:9000  
Server authentication token: .....  
SonarQube authentication token. Mandatory when anonymous access is disabled.

Advanced...  
Delete SonarQube

Add SonarQube  
List of SonarQube installations

**Docker Slaves**  
Docker Provisioner: Compose docker containers

Save Apply





# Add Sonar quality check #3

- Configure sonar
  - Update pipeline allowing sonar analysis

```
pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        echo 'Hello world!'
        echo 'Coucou !'
        echo 'welcome !'
        script {
          docker.withTool('latest') {
            docker.image('maven:3-jdk-8-slim').inside("-v $WORKSPACE:/usr/src/myproject:rw -v $HOME/.m2:/root/.m2:rw -w /usr/src/myproject") { c ->
              sh "mvn clean package"
            }
          }
        }
      }
    }
    stage('Quality check') {
      steps {
        script {
          docker.withTool('latest') {
            docker.image('maven:3-jdk-8-slim').inside("--network ci_cinet -v $WORKSPACE:/usr/src/myproject:rw -v $HOME/.m2:/root/.m2:rw -w /usr/src/myproject") { c ->
              withSonarQubeEnv('sonar') {
                sh 'mvn sonar:sonar'
              }
            }
          }
        }
      }
    }
  }
}
```



# Package the application #1

- Add a stage to package the application
  - Update pipeline allowing packaging

```
def IMAGE_NAME='named-your-image'

pipeline {
    agent any

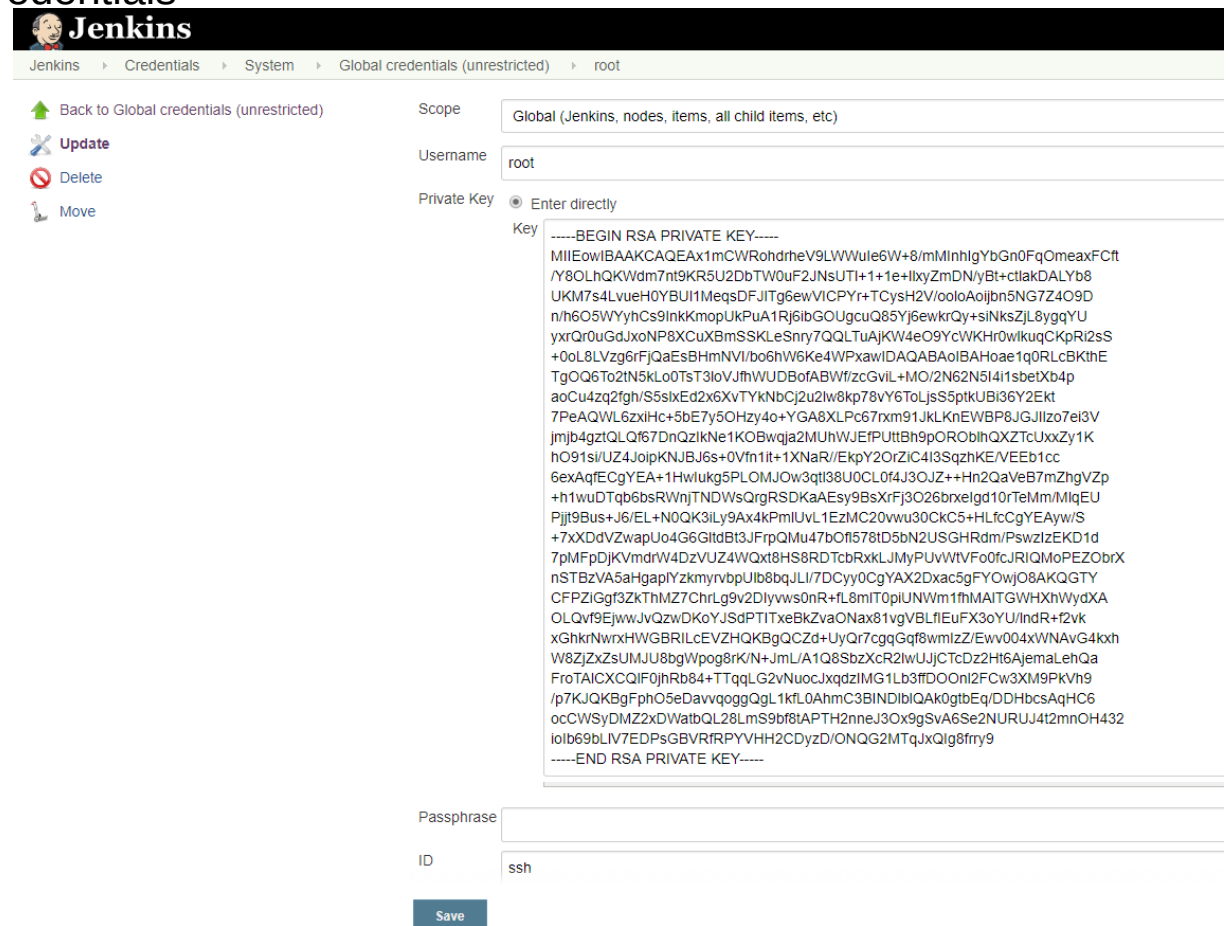
    stages {
        stage('Build') {
            steps {
                echo 'Hello world!'
                echo 'Coucou !'
                echo 'Welcome !'
                script {
                    docker.withTool('latest') {
                        docker.image('maven:3-jdk-8-slim').inside("-v $WORKSPACE:/usr/src/myproject:rw -v $HOME/.m2:/root/.m2:rw -w
/usr/src/myproject") { c ->
                            sh "mvn clean package"
                        }
                    }
                }
            }
        }
        stage('Quality check') {
            steps {
                script {
                    docker.withTool('latest') {
                        docker.image('maven:3-jdk-8-slim').inside("--network ci_cinet -v $WORKSPACE:/usr/src/myproject:rw -v
$HOME/.m2:/root/.m2:rw -w /usr/src/myproject") { c ->
                            withSonarQubeEnv('sonar') {
                                sh 'mvn sonar:sonar'
                            }
                        }
                    }
                }
            }
        }
        stage('Package app') {
            steps {
                script {
                    docker.withTool('latest') {
                        docker.withRegistry('https://registry.uggla.fr/') {
                            def myImg = docker.build("${IMAGE_NAME}:${env.BUILD_ID}")
                            myImg.push()
                        }
                    }
                }
            }
        }
    }
}
```

# Deploy the application #1

- Deploy and run the application
  - Add the Docker registry certificate to vmdev
- Create a ssh root keys
- Load private keys to into Jenkins credentials

REGISTRY\_FQDN=registry.uggla.fr

```
# Get certificate from registry
curl -L http://${REGISTRY_FQDN}:81/ca.crt >
/usr/local/share/ca-certificates/ca-registry.crt
update-ca-certificates
systemctl restart docker.service
```



The image shows the Jenkins web interface for configuring a credential. The breadcrumb trail is: Jenkins > Credentials > System > Global credentials (unrestricted) > root. On the left, there are buttons: 'Back to Global credentials (unrestricted)', 'Update', 'Delete', and 'Move'. The main form has the following fields:

- Scope:** Global (Jenkins, nodes, items, all child items, etc)
- Username:** root
- Private Key:** ☒ Enter directly
- Key:** -----BEGIN RSA PRIVATE KEY-----  
MIIEowIBAAKCAQEAx1mCWrohV9LWWule6W+8/mMinhlGyBt+ctlakDALYb8  
/Y8QLhQKWdm7nt9KR5U2DbTW0uF2JNsUTi+1+1e+lxYzmDN/yBt+ctlakDALYb8  
UKM7s4LvueH0YBUI1MeqsDFJITg6ewVICPYr+TCysH2V/ooolAoijbn5NG7Z4O9D  
n/h6O5WYyhCs9lnkKmpUkPuA1Rj6ibGOUgcuQ85Yj6ewkrQy+siNksZjL8yggqYU  
yxRQr0uGdJxoNP8XCuXBmSSKLeSnry7QQLTuAJKW4eO9YcWKHr0wkuqCKpRi2sS  
+0oL8LVzg6rFJQaEsBHmNVI/bo6hW6Ke4WPxaw/DAQABAoiBAHoae1q0RLcBKthE  
TgOQ6To2tN5kLo0TsT3loVJfhWUDBoFABWf/zcGvIL+MO/2N62N5i4i1sbtXb4p  
aoCu4zq2fgh/S5sIxEd2x6XvTYkNbCj2u2lw8kp78vY6ToLjsS5ptkUBi36Y2Ekt  
7PeAqWL6zxiHc+5bE7y5OHzy4o+YGA8XLPc67rxm91JkLKnEWBP8JJIz07ei3V  
jmjb4gtzQLQf67DnQzlkNe1KOBwqja2MUhWJEfPUItBh9pORObihQXZTcUxxZy1K  
hO91si/UZ4JoipKNJBj6s+0Vfn1it+1XNaR//EkpY2OrZiC4i3SqzhKE/VEEb1cc  
6exAqfECgYEA+1HwluK5PLOMJOW3gti38U0CL0f4J3OJZ++Hn2QaVeB7mZhgvZp  
+h1wuDTqb6bsRWnjTNDWsQrgRSDKaAEsy9BsXrFj3O26brxelgd10TeMm/MiqEU  
Pijt9Bus+J6/EL+N0QK3iLy9Ax4kPmIUvL1EzMC20vWu30CkC5+HLfcGgYEAyW/S  
+7XXDdVZwapUo4G6GltD8t3JfFpQMu47bOfi578D5bN2USGHRdm/PswzIzEKD1d  
7pMfPjDKVmdrW4DzVUZ4WQxt8HS8RDTcbRxxLJMyPUvWVfVf0fCJRlQMPEZObrX  
nSTBzVA5aHgaplYzkmrvbPUBb8bqJLI/7DCyY0CgYAX2Dxacc5gFYOWj08AKQGTy  
CFPZIGf3ZkThMZ7Chrg9v2Dlyvws0nR+Hl8mIT0piUNWm1fhMAITGWHXhWydXA  
OLQv9EjwwJvQzwDKoYJsdPTITxeBkZvaONax81vgVBLfiEuFX3oYU/IndR+f2vk  
xGhkrNwrxHWGBRILcEVZHQBKqQCZd+UyQr7cgqGqf8wmlZZ/Ewv004xWVNAwG4kxh  
W8ZjZx2sUMJU8bgWpog8RkN+JmL/A1Q8SbzXcR2lwUJJCTcdZ2Ht6AjemaLehQa  
FroTAICXCQIF0jhRb84+TTqLg2vNuocJxqdzlMG1Lb3ffDOOnl2FCw3XM9PKvh9  
/p7KJQKbgFphO5eDavvqoggQgL1kfL0AhmC3BNDIblQAK0gtbEq/DDHbcsAqHC6  
ocCWSyDMZ2x2DWatbQL28LmS9bf8tAPTH2nneJ3Ox9gSvA6Se2NURUJ4t2mnOH432  
loib69bLVIEDPsGBVRIRPYVHH2CDyzD/ONQG2MTqJxQlgr8fry9  
-----END RSA PRIVATE KEY-----
- Passphrase:**
- ID:** ssh
- Save:**



## Deploy the application #2

- Deploy and run the application
  - Add Jenkins ssh pipeline steps plugin
- Update Jenkinsfile to add deployment

```
def IMAGENAME='named-your-image'
def VMDEV='188.166.48.108'

pipeline {
    agent any

    ... stage('Deploy app') {
        steps {
            script {
                withCredentials([sshUserPrivateKey(
                    credentialsId:'ssh',
                    keyFileVariable:'pkey',
                    passphraseVariable:'',
                    usernameVariable:'user')]){
                    sh "ssh -i ${pkey} -o StrictHostKeyChecking=no root@${VMDEV} docker run
-i registry.ugla.fr/${IMAGENAME}:${env.BUILD_ID}"
                }
            }
        }
    }
}
}
```

